Министерство образования и науки Российской Федерации

(МИНОБРНАУКИ РОССИИ)

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (ТГУ)

Институт прикладной математики и компьютерных наук

Кафедра защиты информации и криптографии

**КУРСОВАЯ РАБОТА**

БИБЛИОТЕКА ДЛЯ РАБОТЫ С БУЛЕВЫМИ ФУНКЦИЯМИ ДЛЯ ЯЗЫКА ПРОГРАММИРОВАНИЯ LYAPAS

Муругов Михаил Алексеевич

Руководитель

канд. физ.-мат. наук, доцент

\_\_\_\_\_\_\_\_\_\_\_\_И.А.Панкратова «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_201\_\_\_г.

Студент группы № 1155

\_\_\_\_\_\_\_\_\_\_\_\_\_М.А.Муругов

Томск 2018

ОГЛАВЛЕНИЕ

Введение………………………………………………………………………………………….2

1. Описание алгоритмов на математическом языке…………………………………………..3
   1. Принадлежность булевой функции к классу 
   2. Принадлежность булевой функции к классу 
   3. Принадлежность булевой функции к классу монотонных булевых функций
   4. Преобразование Мёбиуса булевой функции
   5. Принадлежность булевой функции к классу линейных булевых функций
   6. Отражение вектора значений булевой функции
   7. Принадлежность булевой функции к классу самодвойственных булевых функций
2. Идеи программных реализаций
   1. Принадлежность булевой функции к классу 
   2. Принадлежность булевой функции к классу 
   3. Принадлежность булевой функции к классу монотонных булевых функций
   4. Преобразование Мёбиуса булевой функции
   5. Принадлежность булевой функции к классу линейных булевых функций
   6. Отражение вектора значений булевой функции
   7. Принадлежность булевой функции к классу самодвойственных булевых функций
3. Экспериментальные данные
   1. Принадлежность булевой функции к классу монотонных булевых функций
   2. Преобразование Мёбиуса булевой функции
   3. Принадлежность булевой функции к классу линейных булевых функций
   4. Отражение вектора значений булевой функции
   5. Принадлежность булевой функции к классу самодвойственных булевых функций
4. Заключение

Список использованных источников и литературы

Приложения

**ВВЕДЕНИЕ**

Целью этой курсовой работы было написание библиотеки для работы с булевыми функциями (определение принадлежности к замкнутым классам, различные преобразования и т.д.) для языка программирования LYaPAS. В дальнейшем планируется, что эта библиотека будет использоваться для реализации криптографических алгоритмов и прочих нужд.

Для криптографии булевы функции важны т.к. они, в частности, используются в качестве комбинирующих и фильтрующих функций при построении поточных шифров; для блочных шифров они используются в качестве функций блоков замены и т.д.

В нынешнее время язык LYaPAS уже выигрывает по скорости на некоторых алгоритмах, но всё ещё требует доработок и улучшений, в следствие чего и был выбран.

В качестве базиса в языке уже реализованы побитовые операции для булевых векторов любой длины, а также для 32-х битных векторов функция подсчёта веса и генерация псевдослучайного вектора. Всё это используется в настоящей работе для реализации более сложных вещей относительно булевых функций.

**ОПИСАНИЕ АЛГОРИТМОВ НА МАТЕМАТИЧЕСКОМ ЯЗЫКЕ**

Все определения (за исключением преобразования Мёбиуса и отражения вектора значений) взяты из [1].

***Принадлежность булевой функции к классу ***

**Определение.** Булева функция *сохраняет константу 0* (*принадлежит классу* *)*, если на наборе из всех нулей функция принимает значение нуль.

Алгоритм:

Вход:  – булева функция

Выход: “ принадлежит классу ?”

Шаг 1) Если , то ответ “Да”

Иначе ответ “Нет”

***Принадлежность булевой функции к классу ***

**Определение.** Булева функция *сохраняет константу 1* (*принадлежит классу* ), если на наборе из всех единиц функция принимает значение единица.

Алгоритм:

Вход:  – булева функция

Выход: “ принадлежит классу ?”

Шаг 1) Если , то ответ “Да”

Иначе ответ “Нет”

***Принадлежность булевой функции к классу монотонных булевых функций***

**Определение.** Булева функция  называется *монотонной* (*принадлежит классу* ), если для любой пары наборов  и  таких, что , выполняется условие .

Алгоритм определения принадлежности булевой функции к классу монотонных булевых функций приведён в [1].

***Преобразование Мёбиуса булевой функции***

**Определение.** *Положительной конъюнкцией* называется элементарная конъюнкция, не содержащая инверсий переменных. Договоримся обозначать положительную конъюнкцию через .

**Определение.** *Полиномом Жегалкина*, или *алгебраической нормальной формой (АНФ)*, булевой функции  называется дизъюнкция с исключением различных положительных конъюнкций переменных из множества , то есть формула вида , задающая функцию .

**Определение.** *Преобразованием Мёбиуса* называется функция , где  – множество всех булевых функций от переменных. С помощью преобразования Мёбиуса решается задача построения АНФ булевой функции, и вычислить его значения для функции  можно по формуле , где .

Довольно быстрый и простой способ преобразования Мёбиуса, который был взят за основу программной реализации, приведён в [2].

Преобразование Мёбиуса связано с полиномом Жегалкина следующим образом: значение  на наборе аргументов говорит о том, есть ли положительная конъюнкция аргументов со значением  из этого набора в АНФ функции  ( – положительная конъюнкция есть,  – положительной конъюнкции нет). Набор аргументов  соответствует константе .

***Принадлежность булевой функции к классу линейных булевых функций***

**Определение.** *Длиной* булева вектора назовем количество его компонент, а *весом* вектора – количество компонент, равных единице

Длину булева вектора  в дальнейшем будем обозначать . Запись , где  – булева функция, будет обозначать длину вектора её значений.

Вес булева вектора  в дальнейшем будем обозначать . Запись , где  – булева функция, будет обозначать вес вектора её значений.

**Определение.** *Длиной полинома Жегалкина* назовем количество конъюнкций в полиноме, а его *степенью* – наибольший из рангов конъюнкций, входящих в полином.

**Определение.** Полином Жегалкина называется *линейным*, если его степень не превышает единицы.

**Определение.** Булева функция называется *линейной* (*принадлежит классу* ), если ее полином Жегалкина линеен.

Определить линейность булевой функции  достаточно просто. Для этого возьмём булеву функцию  и построим по вектору значений  АНФ функции  (связь преобразования Мёбиуса и построения АНФ функции описаны на с. 4), если АНФ линейна, то функция  – линейна.

***Отражение вектора значений булевой функции***

**Определение.** *Отражением* вектора значений булевой функции является обмен значениями функции на противоположных наборах аргументов.

В дальнейшем отражение вектора значений булевой функции  будем обозначать 

***Принадлежность булевой функции к классу самодвойственных булевых функций***

**Определение.** Булева функция  называется *двойственной булевой функции* , если она получена из  инверсией всех аргументов и самой функции, то есть .

**Определение.** Булева функция  *самодвойственна* (*принадлежит классу* ), если она равна двойственной себе функции, то есть .

Алгоритм определения принадлежности булевой функции к классу самодвойственных булевых функций приведён в [1].

**ИДЕИ ПРОГРАММНЫХ РЕАЛИЗАЦИЙ**

Перед изложением дальнейшего материала необходимо кое-что обозначить:

Во-первых, булевы функции в языке LYaPAS представляются векторами их значений.

Во-вторых, вектора значений булевых функций хранятся в логических комплексах L, каждый элемент которого занимает в памяти 4 байта(32 бита). Таким образом, т.к. , то функция до 5 аргументов включительно помещается в один элемент комплекса. От 6 в 2 элемента, от 7 в 4 и т.д. Количество элементов комплекса, необходимых для хранения функции от  аргументов можно вычислить по формуле .

В-третьих, значения булевой функции в памяти хранятся в привычном нам порядке (младшие биты справа, старшие биты слева), при этом нулевой бит нулевого элемента комплекса соответствует значению , следующий за ним  и т.д.

Также обратите внимание, что в этом разделе находятся лишь идеи, сами программные реализации смотрите в приложении.

***Принадлежность к классу ***

Проверка булевой функции на принадлежность к классу  тривиальна. Необходимо просто посмотреть на первый бит вектора её значений. Если этот бит равен нулю, то функция сохраняет константу 0.

***Принадлежность к классу ***

Для проверки принадлежности булевой функции к классу  необходимо посмотреть на старший бит вектора её значений. Если этот бит равен 1, то функция сохраняет константу 1. Но проверка булевой функции на принадлежность к классу  немного сложнее, чем к классу , т.к. у функций, зависящих от  аргументов старший бит вектора значений находится в нулевом элементе комплекса и его сначала необходимо найти. В общем же случае найти старший бит вектора значений функции можно по следующим правилам: , , где  – индекс элемента комплекса, а  – номер бита в элементе с индексом .

***Принадлежность булевой функции к классу монотонных булевых функций***

Т.к. булева функция помещена в «блоки» по 32 бита, то перед стартом рекурсии можно проверить её на монотонность вплоть до 5 компоненты следующими действиями:

L1i < 1 & AAAAAAAAh ⇒ a \*\*\*Проверяем на монотонность на наборах,

a & L1i ⊕ a ↦2 \*\*\*соседних по пятой компоненте

L1i < 2 & CCCCCCCCh ⇒ a \*\*\*Проверяем на монотонность на наборах,

a & L1i ⊕ a ↦2 \*\*\*соседних по четвёртой компоненте

L1i < 4 & F0F0F0F0h ⇒ a

a & L1i ⊕ a ↦2 \*\*\*...

L1i < 8 & FF00FF00h ⇒ a

a & L1i ⊕ a ↦2

L1i < 16 ⇒ a \*\*\*Проверяем на монотонность на наборах,

a & L1i ⊕ a ↦2 \*\*\*соседних по первой компоненте

После того, как каждый элемент комплекса проверен таким образом и немонотонность не обнаружена, то запускается рекурсивная функция, которая проверяет на монотонность по остальным компонентам.

***Преобразование Мёбиуса булевой функции***

Как следует из способа, изложенного в [2] и учитывая, что операция  ассоциативна, преобразование Мёбиуса рекурсивно реализуется по следующему алгоритму:

Вход:  – булева функция

Выход: 

Шаг 1) Разбиваем вектор значений булевой функции на младшую и старшую часть  и  соответственно

Шаг 2) 

Шаг 3) Если , то выход

Иначе выполнить шаги 1-3 для  и 

***Принадлежность булевой функции к классу линейных булевых функций***

Алгоритм на проверку принадлежности булевой функции довольно прост. Необходимо выполнить преобразование Мёбиуса для этой функции и посмотреть, есть ли хотя бы одна единица на наборе аргументов с более, чем одной единицей.

Алгоритм:

Вход:  – булева функция

Выход: “ – линейна?”

Шаг 1) 

Шаг 2) 

Шаг 3) Для :

Шаг 3.1) , где  – набор аргументов с единственной единицей на -ой

позиции

Шаг 4) Если , то ответ “Да”

Иначе ответ “Нет”

Этот алгоритм обусловлен тем, что зануляя значение функции на наборе аргументов из нулей и с одной единицей, мы исключаем из рассмотрения положительные конъюнкции 0 и 1 рангов и если остаётся хоть одна положительная конъюнкций большего ранга, то, как следует из определения линейной функции, она не линейна.

***Отражение вектора значений булевой функции***

Отражение вектора значений булевой функции программно довольно нетривиально, т.к. нет таких средств, которые позволили бы сделать это за одну операцию. В языке LYaPAS, т.к. вектора значений булевых функций разбиты на «блоки» по 32 бита, преобразование выполняется следующим образом: сначала выполняется отражение каждого «блока» по отдельности, затем первый «блок» меняется местами с последним, второй с предпоследним и т.д.

Также, т.к. вектора значений булевых функций от  переменных включительно помещаются в один элемент логического комплекса L, то после отражения таких векторов их необходимо будет побитово сдвинуть вправо на  бита, т.к. после отражения младшие биты станут старшими в 32-х битном «блоке».

Функция, выполняющая отражение 32-х битного «блока»:

reverseBits(a/b)

\*\*\*a – входной вектор

\*\*\*b – отраженный вектор a

a > 1 & 55555555h ⇒ v

a & 55555555h < 1 ∨ v ⇒ b \*\*\*Меняем местами чётные и нечётные биты

b > 2 & 33333333h ⇒ v

b & 33333333h < 2 ∨ v ⇒ b \*\*\*Меняем местами пары битов

b > 4 & 0f0f0f0fh ⇒ v

b & 0f0f0f0fh < 4 ∨ v ⇒ b \*\*\*Меняем местами последовательности из 4-х битов

b > 8 & 00ff00ffh ⇒ v

b & 00ff00ffh < 8 ∨ v ⇒ b \*\*\*Меняем местами байты

b > 16 ⇒ v

b < 16 ∨ v ⇒ b \*\*\*Меняем местами 2-х байтовые слова

\*\*

***Принадлежность булевой функции к классу самодвойственных булевых функций***

Программно проверка на принадлежность булевой функции к классу самодвойственных булевых функций реализована согласно следующему алгоритму:

Вход:  – булева функция

Выход: “ – самодвойственна?”

Шаг 1) Разбиваем вектор значений булевой функции на младшую и старшую часть  и  соответственно

Шаг 2) Если , то ответ “Да”

Иначе ответ “Нет”

 – этим самым действием мы сопоставляем значения функции на противоположных наборах аргументов, а затем проверяем условие .

**ЭКСПЕРИМЕНТАЛЬНЫЕ ДАННЫЕ**

Далее представлены графики зависимости времени работы функций над булевыми функциями от количества переменных в булевых функциях. Для каждого количества переменных выполнялось по 100 итераций и на вход подавался наихудший случай (проверка констант на монотонность и т.д.).

Для проверки принадлежности к классам  и  эксперименты не проводились, т.к. сложность этих операций  и смотреть зависимость времени работы функций от количества переменных в булевой функции бессмысленно.

***Принадлежность булевой функции к классу монотонных булевых функций***

Как видно из графика, при  функция имеет сложность .

***Преобразование Мёбиуса булевой функции***

Как видно из графика, при  функция имеет сложность , но даже для функции от 31 аргумента выполняется быстрее, чем за секунду.

***Принадлежность булевой функции к классу линейных булевых функций***

Как видно из графика, при  функция имеет сложность .

***Отражение вектора значений булевой функции***

Как видно из графика, при  функция имеет сложность .

***Принадлежность булевой функции к классу самодвойственных булевых функций***

Как видно из графика, при  функция имеет сложность .

**ЗАКЛЮЧЕНИЕ**

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ**

1. Быкова С.В. Учебно-методический комплекс «Булевы функции». Томск 2006.
2. Панкратова И.А. Учебное пособие «Булевы функции в криптографии». Томск 2014.

**ПРИЛОЖЕНИЯ**

Во всех функциях L1, L2 – вектора значений булевых функций,  – количество переменных, от которых они зависят.

\*\*\*Функция копирования булевой функции от  аргументов из комплекса L1 в \*\*\*комплекс L2 (используется, т.к. операция L1⇒L2 не работает на функциях от \*\*\*большого количества аргументов)

copyBF(L1,n/L2)

In + 31 > 5 ⇒ Q2

Oi

§1 L1i ⇒ L2i

∆i

↑(i<Q2)1

\*\*

\*\*\*Рекурсивная функция для выполнения преобразование Мёбиуса

\*\*\*(вспомогательная, должна вызываться ТОЛЬКО из MobiusBF(/))

MobiusBFHelper(L1,l,h/L1)

\*\*\*l - начальный индекс для преобразования

\*\*\*h - конечный индекс для преобразования

\*\*\*Преобразования выполняется для [l,h) элементов

h - l > 1 ↪2 + l ⇒ j ⇒ m \*\*\*Проверяем условие выхода и находим индекс середины L1

l ⇒ i

§1 L1i ⊕ L1j ⇒ L1j

∆i ∆j

↑(j<h)1

\*MobiusBFHelper(L1,l,m/L1)

\*MobiusBFHelper(L1,m,h/L1)

§2 \*\*

\*\*\*Преобразование Мёбиуса булевой функции

MobiusBF(L1,n/L2)

\*copyBF(L1,n/L2)

In + 31 > 5 ⇒ q ⇒ Q2 \*\*\*Считаем, сколько элементов в L1

Oi

§1 \*\*\*Выполняем преобразование Мёбиуса для каждого элемента L2

L2i < 1 & AAAAAAAAh ⊕ L2i ⇒ L2i

L2i < 2 & CCCCCCCCh ⊕ L2i ⇒ L2i

L2i < 4 & F0F0F0F0h ⊕ L2i ⇒ L2i

L2i < 8 & FF00FF00h ⊕ L2i ⇒ L2i

L2i < 16 ⊕ L2i ⇒ L2i

∆i ↑(i<q)1

q ⊕ 1 ↦2 \*\*\*↑(q≠1)2

32 - In ⇒ n

L2.0 < n > n ⇒ L2.0 \*\*\*”Отсекаем” лишние биты

→3

§2 \*MobiusBFHelper(L2,0,q/L2) \*\*\*Вызов рекурсивной функции

§3 \*\*

\*\*\*Проверка булевой функции на линейность

isBFLinear(L1,n/f)

Of

S1 ⇒ s

@+L2(s) \*MobiusBF(L1,n/L2)

L2.0 & FFFFFFFEh ⇒ L2.0 \*\*\*Зануляем бит, соответствующий набору аргументов из \*\*\*нулей

Oi

§1 \*\*\*цикл зануления битов, соответствующих наборам аргументов с одной единицей

Ii > 5 ⇒ k \*\*\*Находим индекс нужного элемента

Ii & 31 ⇒ s \*\*\*Находим номер нужного бита

Is ¬ & L2k ⇒ L2k \*\*\*Зануляем нужный бит

∆i

↑(i<n)1

In + 31 > 5 ⇒ q \*\*\*Вычисляем, сколько необходимо элементов для хранения булевой

\*\*\*функции

Oi

§2 L2i % ↦3 \*\*\*Проверяем на наличие единиц на остальных наборах аргументов

∆I ↑(i<q)2

∆f \*\*\*Если единиц не обнаружили, то функция линейна

§3 \*\*

\*\*\*Принадлежность к классу 

isBFT0(L1,n/f)

L1.0 & 1 ⊕ 1 ⇒ f

\*\*

\*\*\*Принадлежность к классу 

isBFT1(L1,n/f)

Of

In + 31 > 5 ⇒ i \*\*\*Индекс элемента комплекса

In - 1 & 31 ⇒ j \*\*\*Номер бита

L1i & Ij ↪1

∆f

§1 \*\*

\*\*\*Отражение 32-х битного вектора

reverseBits(a/b)

\*\*\*a – исходный 32-х битный вектор

\*\*\*b – отражённый вектор a

a > 1 & 55555555h ⇒ v

a & 55555555h < 1 ∨ v ⇒ b \*\*Меняем местами чётные и нечётные биты

b > 2 & 33333333h ⇒ v

b & 33333333h < 2 ∨ v ⇒ b \*\*\*Меняем местами пары битов

b > 4 & 0f0f0f0fh ⇒ v

b & 0f0f0f0fh < 4 ∨ v ⇒ b \*\*\*Меняем местами последовательности из 4-х битов

b > 8 & 00ff00ffh ⇒ v

b & 00ff00ffh < 8 ∨ v ⇒ b \*\*\*Меняем местами байты

b > 16 ⇒ v

b < 16 ∨ v ⇒ b \*\*\*Меняем местами 2-х байтовые слова

\*\*

\*\*\*Отражение вектора значений булевой функции

revBF(L1,n/L2)

OQ2

In + 31 > 5 ⇒ q ⇒ j \*\*\*Считаем, сколько элементов требуется для хранения вектора \*\*\*значений функции

∇j \*\*\*Индекс самого старшего элемента

§1 L1j @>L2 \*\*\*Переписываем элементы L1 в L2 в обратном порядке

∇j ↑(j<q)1

q ⊕ 1 ↪3

Oj

§2 \*reverseBits(L2j/b)

b ⇒ L2j \*\*\*Отражаем каждый элемент комплекса L2

∆j ↑(j<Q2)2

→4

§3 32 - In ⇒ s

\*reverseBits(L2.0/b) \*\*\*Если вектор значений помещается в один элемент,

b > s ⇒ L2.0 \*\*\*То отражаем его и сдвигаем на нужную позицию

§4 \*\*

\*\*\*Принадлежность к классу линейных булевых функций

isBFSelfdual(L1,n/f)

Of ∆f

In + 31 > 5 ⇒ q

q ⊕ 1 ↪3 \*\*\*↑(q=1)3

∇q Oi

§1 \*reverseBits(L1q/b) \*\*\*Проверяем, не совпадают ли значения функции

b¬ ⊕ L1i ↦2 \*\*на противоположных наборах аргументов

∆i ∇q ↑(i<q)1

→4

§2 Of

→4

§3 \*reverseBits(L1.0/b)

32 - In ⇒ s

b¬ > s ⊕ L1.0 ↦2

§4 \*\*

\*\*\*”Эквивалентны ли булевы функции?”

isBFEquals(L1,L2,n/f)

Of ∆f

In + 31 > 5 ⇒ n

Oi

§1 L1i ⊕ L2i ↦2

∆I ↑(i<n)1

→3

§2 Of

§3 \*\*

\*\*\*Генерация константы 0

BFconst0(n/L1)

In + 31 > 5 ⇒ Q1 \*\*\*Вычисляем количество элементов, необходимых для храненя

OL1 \*\*\*функции и обнуляем её (обнуление происходит по мощности)

\*\*

\*\*\*Генерация константы 1

BFconst1(n/L1)

In + 31 > 5 ⇒ Q1

⁻L1

Q1 ⊕ 1 ↦1 \*\*\*↑(Q1≠1)1

32 - In ⇒ s \*\*\*”Отсекаем” лишние биты, если есть необходимость

L1.0 < s > s ⇒ L1.0

§1 \*\*

\*\*\*Генерация псевдослучайной булевой функции

genBF(n/L1)

OQ1

In + 31 > 5 ⇒ q

§1 X > 16 ⇒ a \*\*\*Операция X используется дважды, т.к. младшие биты,

X ⊕ a @>L1 \*\*\*выдаваемые ею, не совсем случайны

↑(Q1<q)1

Q1 ⊕ 1 ↦2 \*\*\*↑(Q1≠1)2

32 - In ⇒ n

L1.0 > n ⇒ L1.0

§2 \*\*

isBFMonotonicHelper(L1,l,h/f)

\*\*\*Рекурсивная функция для определения монотонности булевой функции

\*\*\*Должна вызываться ТОЛЬКО из isBFMonotonic(/)

Of

h - l > 1 + l ⇒ m ⇒ j

l ⇒ i

§1 L1i & L1j ⊕ L1i ↦3

∆i ∆j ↑(j<h)1

∆f

h - l ⊕ 2 ↪3 \*\*\*↑((h-l)=2)3

\*isBFMonotonicHelper(L1,l,m/f)

f ↪3

\*isBFMonotonicHelper(L1,m,h/f)

§3 \*\*

\*\*\*Проверка булевой функции на монотонность

isBFMonotonic(L1,n/f)

In + 31 > 5 ⇒ n

Oi Of

§1 L1i < 1 & AAAAAAAAh ⇒ a

a & L1i ⊕ a ↦2 \*\*\*Проверяем на монотонность на наборах, соседних по пятой \*\*\*компоненте

L1i < 2 & CCCCCCCCh ⇒ a

a & L1i ⊕ a ↦2 \*\*\*Проверяем на монотонность на наборах, соседних по

\*\*\*четвёртой компоненте

L1i < 4 & F0F0F0F0h ⇒ a

a & L1i ⊕ a ↦2 \*\*\*...

L1i < 8 & FF00FF00h ⇒ a

a & L1i ⊕ a ↦2

L1i < 16 ⇒ a

a & L1i ⊕ a ↦2 \*\*\*Проверяем на монотонность на наборах, соседних по первой

\*\*\*компоненте

∆i ↑(i<n)1

∆f

n ⊕ 1 ↪2

\*isBFMonotonicHelper(L1,0,n/f)

§2 \*\*

\*\*\*Функция подсчёта веса булевой функции

BFweight(L1,n/q)

Oq

In + 31 > 5 ⇒ n

n ⊕ 1 ↪3

Oi

§1 L1i % + q ⇒ q

∆I ↑(i<n)1

→3

§2 32 - In ⇒s

L1.0 < s > s % ⇒ q

§3 \*\*

\*\*\*Вывод значения 32-х битной переменной в двоичном виде

\*\*\*(младшие биты справа, старшие слева)

printBool(a/)

@+F1(32) OQ1

§1 a & 1 + '0' @>F1.0

a > 1 ⇒ a ↦1

↑(Q1=32)3

§2 '0' @>F1.0

↑(Q1<32)2

§3 /F1>C \*\*

\*\*\*Вывод вектора значений булевой функции

\*\*\*(Младшие биты справа, старшие слева)

printBF(L1,n/)

↑(n>4)4

@+F2(16) OQ2

32 - In ⇒ s

L1.0 < s > s ⇒ a

§1 a & 1 + '0' @>F2.0

a > 1 ⇒ a ↦1

In ⇒ n

↑(Q2=n)3

§2 '0' @>F2.0

↑(Q2<n)2

§3 /F2>C

→6

§4 In > 5 ⇒ n ⇒ i

∇i

§5 \*printBool(L1i/)

∇i

↑(i<n)5

§6 \*\*